

A CONTROLLER FOR A MEMORY SYSTEM HAVING MULTIPLE PARTITIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

5 FIELD OF THE INVENTION

The present invention relates generally to memory controllers in a computing system and more particularly to a controller for a memory system in which the storage array is partitioned into a number of independent sections.

10 DESCRIPTION OF THE RELATED ART

In current graphics subsystems, the speed and number of graphical processing elements has increased enough to make the graphics memory subsystem a barrier to achieving the higher performance that these elements may be capable of achieving. Typical graphics processing elements which are the memory clients include the host processor, a texture engine, a z-buffer engine, a color engine, a 2D-graphics engine, a 3D-graphics engine and a display engine.

FIG. 1 shows a graphics memory controller of the prior art. The memory controller 10 acts as a switch to determine which of several graphics processing clients 12, 14, 16, 18, 20 can access the memory storage array 22, which is organized as a single, i.e., monolithic, partition. Each client requests one or more cycles of the memory storage array 22 which transfers, in each cycle, a data quantity equal to the size of the data bus of the array.

Monolithic memory subsystems for the various graphics clients have evolved in an attempt to overcome this barrier chiefly by using a wider memory data bus for increased throughput. Memory busses in current architectures are now 128 bits physically, and 256 bits (32 bytes), effectively, when the minimum data transfer requires both phases of single clock cycle. (Hereinafter, when referring to the size of the data bus, the effective size, rather than physical size is meant.) The size of the memory data bus sets the size of the minimum access that may be made to the graphics memory subsystem and for some devices that make use of the graphics memory, 32 bytes is an acceptable minimum.

However, for many devices in a graphics subsystem, a minimum size of 32 bytes is inefficient. As geometry objects become smaller and finer, a minimum access of 32 bytes transfers more data than is needed by the various graphics engines for the geometry object. One

measure of fetching efficiency of the access is the ratio of pixels used for the geometry object to pixels fetched during an access. As the size of the memory bus increases or the minimum access increases, this ratio becomes smaller. A small ratio implies a large amount of wasted memory throughput in the graphics memory subsystem.

5 FIG. 2 shows a plot of the fetching efficiency of the prior art system of FIG. 1, in which the fetching efficiency is plotted on the vertical scale and the relative number of pixels per triangle is plotted on the horizontal scale. The plot shows that, as the number of pixels per triangle decreases, the fetching efficiency also decreases to a point that becomes unacceptably low for small triangles, i.e., triangles having approximately 10 to 100 pixels. At that point more
10 than half the data fetched is not used for the triangle.

Thus, it is desirable to improve the fetching efficiency to the local memory storage but without altering the view to the memory clients of memory as a single unit to maintain compatibility with existing memory clients.

Monolithic memory subsystems are also inefficient in another way. Because the topology of the array is such that the entire or substantial part of the address path and the control path must reach each device that makes up the memory array, the electrical load on these paths is quite high. This leads to a slower cycle time for the array which translates into a loss of throughput than the array can supply. A partial answer to this problem is the replication and buffering of the address and control paths to portions of the array, which has the effect of increasing the cost of
20 the array to gain a portion of the lost performance.

Therefore, there is a need to improve the efficiency of the memory access and the topology of the array without changing the existing memory clients. Improved efficiency and topology would lead to higher throughputs from the memory array and would lead to better use of the available memory throughput by the graphics subsystem thereby improving the
25 performance of the graphics subsystem without a major impact on the memory clients.

BRIEF SUMMARY OF THE INVENTION

The present invention meets the above need. A system in accordance with the present invention includes a plurality of memory partitions each operable to service a memory request
30 independently of other memory partitions. The memory request includes information specifying a location of requested data in the memory subsystem and a data transfer size for the request. A

plurality of client queues, one for each memory client and each partition is also included along with a plurality of routing circuits, wherein one routing circuit is connected to each memory client and to the client queues. Each routing circuit is operative to determine for each request, independently of the other requests, one or more partitions needed to service the request based on the specified location and transfer size, and each routing circuit is operative to route and store, independently of the other requests, each of the requests in the client queues of the one or more servicing partitions. Further included are a plurality of arbiter circuits, one arbiter circuit connected to each partition and to the client queues for the partition, where each arbiter circuit is operative to select, independently of other partitions, based on a priority policy, a request for servicing from one of the client queues for the partition, and to transfer each of the selected requests, independently of the other selected requests, to the servicing partition.

A method of servicing memory requests from at least one memory client in accordance with the present invention includes the following steps. First, at least one memory request from the at least one client is received, where the memory request is directed to a memory subsystem having a plurality of independently operable partitions and includes information specifying a location of requested data in the memory subsystem and a data transfer size for the request. Next, one or more partitions needed to service the request based on the specified location and transfer size are determined and the request is routed to each of the one or more servicing partitions. Following this, the routed request is serviced at each servicing partition independently of the other servicing partitions.

One advantage of the present invention is that, for a given memory size, each partition comprises a smaller number of devices. This reduces or eliminates the need for buffered and replicated address and control paths to the devices thereby improving the topology and cycle time of the partition and therefore the memory system.

Another advantage of the present invention is that, if a client requires only a small transfer, then only a few partitions, perhaps a single partition, of the total memory array is involved in the transfer. This leaves the other partitions of the memory array free to perform other small transfers, thus improving the deliverable throughput of the array. Larger transfers are still possible by splitting the request into a number of smaller requests and submitting the multiple requests to the individual partitions for service.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features, aspects and advantages of the present invention will become better understood with regard to the following description, appended claims, and accompanying drawings where:

5 FIG. 1 shows a graphics memory controller of the prior art;

FIG. 2 shows the fetch efficiency of the prior art;

FIG. 3 shows a high-level block diagram of a system in accordance with the present invention;

10 FIG. 4 shows the partitioning of the memory in which the memory as linear array of bytes is mapped to the partitions in an interleaved fashion;

FIG. 5A shows a block diagram for the address, command and write data path for the memory system of the present invention;

FIG. 5B shows one implementation of the router of FIG. 5A;

FIG. 5C shows another implementation of the router of FIG. 5A;

FIG. 6A shows a block diagram for the read data path in accordance with one version of the present invention;

FIG. 6B shows a block diagram for the read data path in accordance with one version of the present invention with partition-size interface hardware;

20 FIG. 6C shows an implementation of the interface hardware included in the multiplexer circuitry of FIG. 6B;

FIG. 6D shows another implementation of the interface hardware included in the multiplexer circuitry of FIG. 6B;

FIG. 6E shows another block diagram for the read data path in accordance with another version of the present invention with partition-size interface hardware;

25 FIG. 6F shows an implementation of the interface hardware included in the multiplexer circuitry of FIG. 6E;

FIG. 6G shows another implementation of the interface hardware included in the multiplexer circuitry of FIG. 6E;

FIG. 6H shows an implementation of the read data path using sequence counts;

30 FIG. 7 shows an alternative embodiment of the command and write data path having an arbiter that permits multiple clients to share a router and client queues;

FIG. 8 shows another alternative embodiment of the command and write path in which the arbitration for access to a partition is in two parts;

FIG. 9 shows yet another alternative embodiment of the command and write path in which the translation block for requester is removed; and

FIG. 10 shows the degree of improvement of the present invention over the prior art.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 3 shows a high-level block diagram of a system in accordance with the present invention. The figure shows that the memory array 24 has an number of independently operable partitions 26a, 26b, 26c 26d, each with a bus 28a-28d and a bus 27a-27d having a width w that is preferably a smaller transfer size than the single prior art bus 15 in FIG. 1. In one embodiment, there are four independent partitions P0-P3 26a, 26b, 26c 26d each with a bus one quarter the size of the non-partitioned bus, i.e., each with a 64 bit bus. Each of the memory system clients 12-20 is connected as before to the memory controller 30 which presents a non-partitioned view of the memory to the clients. The memory controller 30 includes a number of queues 32, 34, 36, 38, 40, 42, 44, 46 that connect to the memory array of each partition and control logic (not shown in FIG. 2) that determines the one or more partitions to which a request should be routed and the one or more partitions from which a response (read data) to a request should be obtained to maintain the appearance of a non-partitioned memory for the clients. Additionally, the control logic in the memory controller arbitrates among the various clients according to a priority assigned to each of the clients.

Partitioning of the memory is shown in FIG. 4 and involves mapping a linear array of bytes to the partitions in an interleaved fashion. In particular, ranges of byte addresses in the linear array are mapped to the various partitions 26a-26d. Preferably, the mapping is such that the lower address bits are mapped to a particular partition so that the memory load is approximately evenly balanced over the partitions. In one embodiment, each partition is mapped to an 8 byte range of memory locations. In a different embodiment, each partition is mapped to a 16 byte range of memory locations but any size mapping can be used without departing from the spirit and scope of the invention. A memory access request can start on any partition and end on any partition as long as it does not wrap around and touch the starting partition. If the transfer

size of a request is so large that it would violate this rule, then the request, itself, is broken up to multiple requests each satisfying the rule.

If the bus size of a partition is smaller than the range of locations mapped to that partition then multiple accesses are made of the partition until the range of locations in the linear map for that partition is satisfied. For example, if a partition is mapped to a range of locations spanning 16 bytes, but the data bus for the partition is only 8 bytes, then two accesses are made of the partition.

In operation there are two cases to consider. In the first case, a client makes a request that maps to exactly one partition in size and alignment. This case involves only one partition and some clients are dedicated to only one partition to simplify accesses for that client. In the second case, a client makes a request that spans two or more partitions. This requires that the request be split up and routed to two or more partitions for servicing of the request. The request can start at any partition. In the subcase of a write request, the write data are split and the write data and write request are routed to the various partitions. Once the write is retired in the arrays for the affected partitions, the write is completed. In the subcase of a read request, the read request is routed to the proper partitions and as the read request is completed by each of the affected partitions the read data is collected and returned to the client.

The partition data bus width and client data bus width may be different. A client with a wider data bus than that of the partition may consume several clocks of partition transfer time. Alternatively, a client with a data bus width that is smaller than that of a partition can request a burst having a size that requires only one clock cycle of the partition to access the data from that partition but requires many clock cycles to multiplex the partition's data onto the smaller client data bus. The read circuitry for the narrow client is configured to use several clocks to multiplex the data from the partition onto the narrow client bus. This ensures that unused data is not fetched, and isolates the low bandwidth client from the high bandwidth memory system.

FIG. 5A shows a block diagram for the address, command and write data path for the memory system of the present invention. For convenience, four memory partitions 26a, 26b, 26c, 26d are shown, but any number of partitions is contemplated for the present invention. Each partition has an arbiter, 56a, 56b, 56c, 56d, having an output that supplies requests to the partitions and a number of inputs, each of which connects to a client queue 62-66, 70-74, 78-82, 86-90. Each arbiter 56a-56d prioritizes the servicing of requests at its inputs according to a

priority policy and each client queue 62-66, 70-74, 78-82, 86-90 that is connected to an arbiter input receives a memory partition request derived from the one or more memory clients 100, 102, 104, such as the various graphics engines mentioned above. While the figure shows three clients, any number of memory clients is possible as long as the necessary hardware is provided for each client.

Priority policies for the arbiters 56a-56d include a least-recently used policy, a fixed priority policy or a round-robin policy or a static or dynamic set of policies depending on the type and nature of the client requests in the client queues. Requests for time critical processes such as display updates or other isochronous processes are given higher priority over requests that have less strict timing requirements.

Any number of clients can request memory operations of the partitions; three clients are shown for convenience. For each client, there is a router 110, 112, 114 that determines, based on an address and reference size contained in the client's request, the one or more partitions 26a-26d, that are needed to service the request. If, in the example shown, each partition 26a-26d, has an effective address map width of $W \times 8$ bits (W bytes) then the router 110-114 decodes a portion of the address in the request to determine which of the W -byte partitions should service the request. In addition to the decoding function of the router, the router, in some versions of the invention, also performs address mapping or address translation functions if they are required for a particular application. The router can also perform the function of breaking up a request that is so large it violates the no-wrap around rule. The router then issues requests to the client queues each of which meets the no-wrap rule.

Memory clients 100-104 can place a partition request into one of the client queues 62-66, 70-74, 78-82, 86-90 independently of any other client placing a partition request into a different queue. This prevents memory clients from blocking or interfering with each other at least insofar as memory operations are involved. This means that no partition waits on another partition before servicing a memory request. This arrangement improves memory system throughput for memory reads and writes that require fewer than an integer number of all of the partitions. For example, if there are four partitions, then requests requiring 1, 2, or 3 or 5, 6, or 7 partitions receive improved service. Requests that require an integer number of all four partitions incur no loss in performance in the partitioned memory system.

FIG. 5B shows one implementation of logic present in the router of FIG. 5A. The router passes through the request and its address and transfer size information and derives the starting partition information by means of a starting partition logic block 116. The starting partition and size are used by the router to determine which partition client queue to load with a request.

FIG. 5C shows another implementation of logic present in the router of FIG. 5A. In this case, the router passes through the request and its address and transfer size information and derives not only the starting partition information but also a sequence count by means of sequence counter 118. Sequence counter 118 is incremented for each new request as long as there is a not busy condition according to gate 119. The starting partition and size are used by the router to determine which partition client queue to load with a request.

FIG. 6A shows a block diagram for the read data path in accordance with one version of the present invention. In this case, the partition data bus width is the same as the client's data bus width. Read requests that touch only a single partition and burst read requests that span a number of partitions must be serviced by those partitions and the read data must be presented in the order requested by the client back to the client. To handle this case and to prevent deadlocks, the hardware of FIG. 6A is required for each client.

For a particular client, each partition in FIG. 6A has a read queue 120a-120d that captures the read data from the partition array 26a-26d. This read queue 120a-120d is controlled by a queue controller 122, 124, 126, 128 that is connected to the read queue 120a-120d, the partition 26a-26d, and to a control block 130. Each of the read queue controllers 122-128 receives a data_valid signal from the partition 26a-26d and provides a queue control signal, q_empty, to the control block 130. Each of the read queues 120a-120d connects to the input of a multiplexer 132 whose output provides the read data to the client to which this circuitry is dedicated. Another queue, called a side queue 134, receives information regarding the starting partition for the read request, sub request ID, and the burst size, if any, for the read request. Sub-requestors 153, 155, 157, in FIG. 7 require sub request IDs because such sub clients share a single request path. In FIG. 6A, control block 130 is connected to the multiplexer 132 to selectively couple one of the inputs to the output of the multiplexer as the data becomes available at the output of the read queue 120a-120d, respectively connected to each multiplexer input. The control block also provides signals to the queue controllers for use in controlling the queues and provides the data valid signals, data_valid_req0a, data_valid_req0b, data_valid_req0c, for each of the clients.

The hardware according to the block diagram of FIG. 6A, operates to collect the various parts of read data that are supplied by the partitions 26a-26d in response to a read request. The partition loads the read data into read queues corresponding to the client that issued the read by using the data_valid signal as a load signal for the client read queue. For example, if the read request starts at partition 1, 26b as determined by the data in the side queue 134 that is received by the control block 130, then when the queue controller 122-128 for a partition 26a-26d discovers that data is ready in the read queue for that partition by means of the q_empty signal becoming false, the control block 130 sets the multiplexer 132 to output the read queue for partition 1, 26b to the particular client. Next, the queue controller for partition 2, 26c is checked to see if the data is ready from partition 2. If so, then the multiplexer 132 is set to output the read queue for partition 2 to the client. This sequence continues until the full amount of data requested by the client is sent to the client. If the data bus size of a partition is smaller than the range of locations mapped to that partition then multiple read data of the partition may be returned until the range of locations in the linear map for that partition is satisfied.

In some cases a memory client can accept data in an out-of-order fashion. This means that instead of accepting data, lowest byte first for example, the client can accept data in a number of other pre-specified orders. In these cases, the queue controllers 122-128 operate to cause the queues 120a-d to output data in a non-first-in-first-out order in order to provide read data in the particular order in which the clients is accepting the data. This increases throughput of the memory system because a client that can receive data out-of-order can accept data as soon as it is available from a partition.

The embodiment of FIG. 6H does not use a side queue, but instead provides each client with a sequence number counter or equivalent mechanism, which increments for each reference that is issued by the client. The sequence count is conveyed to all servicing partitions along with start partition, transfer size and sub request ID if there are multiple clients sharing the same router as shown in FIG. 7. The sequence count is passed through the partition and loaded into the read queues 120a-d with the read data when that partition's read return data returns from memory. The outputs of read queues 120a-d transfer this information to controller 130. Controller 130 examines the partition read queue outputs and selects read data tagged with the next sequence number only after exhausting read data tagged with the current sequence count.

FIG. 6B shows a block diagram for the read data path in accordance with one version of the present invention with partition-size interface hardware, which is needed when the data bus width of a client is different than that of the partition. The data bus width of a partition may be smaller than that of a client, such that the partition must be cycled more than once to obtain enough data to match the bus width of the client. In this case the control logic operates to cause a burst of data (several data words) to be produced from the partition. Interface hardware blocks 140, 142, 144, 146 for holding the burst data are additionally included in the multiplexer circuitry 133. The multiplexer circuitry selects one of the interface blocks to provide some or all of the burst data.

FIG. 6C shows an implementation of the interface hardware included in the multiplexer circuitry of FIG. 6B. This circuitry adapts a W wide partition to a client bus having a width of $2W$. A register 150 and multiplexer 152 capture and hold a portion of the burst data and the remaining portion of the burst data is provided by the associated read queue. In other embodiments, both W -wide paths are provided by registers.

FIG. 6D shows another implementation of the interface hardware included in the multiplexer circuitry of FIG. 6B. In this implementation, the partition bus is W bytes wide and the client bus is $\frac{1}{2} W$ wide. A multiplexer 152 adapts the wider bus to the smaller bus.

FIG. 6E shows another block diagram for the read data path in accordance with another version of the present invention with partition-size interface hardware. In this case, the client bus width is larger than the partition address map size. In this example, there are two interfacing blocks included within the multiplexer circuitry 133. The first interface block 141 is connected to receive data from read queues 120a and 120b and the second interface block 143 is connected to receive data from read queues 120c and 120d.

FIG. 6F shows an implementation of the interface hardware included in block 143 P of the multiplexer circuitry of FIG. 6E. The interface hardware simply passes through the two W -byte sized busses to provide a bus that is $2W$, because the outputs of a pair of read queues are available to the multiplexer circuitry.

FIG. 6G shows another implementation of the interface hardware included in the multiplexer circuitry of FIG. 6E. The interface block shown adapts a partition bus having a width W and address map width $2W$ to a client bus that has a width $4W$. A pair of registers, 154a,

154b, and a pair of multiplexers, 156a, 156b, are used to capture multiple data items from the pair of read queues to which each of the interface block, 141, 143 is connected.

FIG. 7 shows an alternative embodiment of the command and write data path. In this embodiment, an arbiter 151 is added to prioritize a number of sub-requestors 153, 155, 157 that share a command and write data path in order to reduce the hardware that would otherwise be needed for each requester. Each sub-requestor has a sub-request ID. The data_valid signals of FIGs. 6A and 6B and the sub-request ID indicate which of sub-requestors 153, 155, 157 should collect the read data.

FIG. 8 shows another alternative embodiment of the command and write path in which the arbitration for access to a partition is in two parts, arbiter-0a 56a-56d, and secondary arbiter-0b 160-166, and there is a preferential path for a high priority requester 106, which, in the figure, is Req3. Low priority requesters arbitrate to gain access to the partitions using first the arbiter-0a and then secondary arbiter-0b circuitry but the high priority request via client queues 60, 68, 76, 84 arbitrates to gain access using only the arbiter-0b circuitry 160-166. It is also advantageous to introduce a refresh requestor 170 as shown and memory device initialization requests into the arbiter-0b 160-166 blocks.

The above describes the cases in which a (i) processing request is sent to a client, (ii) the client performs the requested processing and makes memory requests and (iii) the memory system routes the memory requests to the proper client queues. However, if a memory client is itself partitioned into an number of processing stages, an alternative is (i) to determine the one or more processing stages required for a processing request, (ii) route the processing requests to the proper stage for processing and then (iii) send the request directly to the appropriate client queue which is connected to a specific partition. A single memory client that is not partitioned into stages may also be restricted to only access data in a single partition and connect directly only to that partition. FIG. 9 shows this alternative embodiment of the command and write path in which there is no router for client 3 106 because the client directs its requests via client queue 84 only to a particular partition, partition 3, 26d. For these partitioned clients, such as a raster-operation pipeline, this simplifies and speeds up the access to the memory subsystem. In addition, the read path is simplified because the read data is directly returned to the requester from the dedicated partition. The multiplexer and control circuitry of FIG. 5 is not required for a requester that is dedicated to a particular partition.

